An Adaptive Mesh Refinement Algorithm for Porous Media Flows [1]

Richard M. Propp and Phillip Colella
Lawrence Berkeley National Laboratory, Berkeley, California 94720

E-mail: RMPropp@lbl.gov

Flows through porous media are characterized by localized phenomena such as fronts. To concentrate computational effort around these localized phenomena, we use Adaptive Mesh Refinement (AMR) techniques developed by Berger and Oliger. We refine in space using a nested hierarchy of block-structured grids. We refine in time by using subcycling – we advance finer grids several times and synchronize them with the coarser grids. We introduce three new innovations in our adaptive algorithm. First, we use a volume-discrepancy method to correct for the lack of freestream preservation at coarse/fine interfaces. Second, we introduce a lagged correction scheme for coarse/fine boundary conditions to minimize the number of multilevel elliptic solves. Finally, we determine refined regions using an estimate of the local truncation error; this is a generic method of tagging cells that is suitable for the hyperbolic-elliptic problem that we are solving. Our results demonstrate that the adaptive mesh refinement algorithm is able to reproduce the results from a single-grid code with a substantial savings in memory and computational effort.

Key Words: adaptive mesh refinement, porous media flows, Godunov methods

1. INTRODUCTION

Many physical processes involve multiphase flow through a porous media. Applications include such diverse areas as petroleum reservoir simulations, groundwater remediation, chemical and materials processing, and biofluids [13].

Flow through a porous media is typically characterized by localized phenomena such as fronts moving through the domain. In addition, there can be heterogeneities in the medium which cause channeling. Clearly, it is desirable to use a computa-

[1]Research supported at UC-Berkeley by the US Department of Energy Mathematical, Information, and Computational Sciences Division, Grants DE-FG03-94ER25205 and DE-FG03-92ER25140; and at the Lawrence Berkeley National Laboratory by the US Department of Energy Mathematical, Information, and Computational Sciences Division, Grant DE-AC03-76SF00098. The first author was also supported by the Computational Sciences Graduate Fellowship Program of the Office of Scientific Computing in the Department of Energy.

tional mesh that can resolve these phenomena. However, since the location of these phenomena may not be known a priori and may change in time, we can not use a static fine mesh around the phenomena. Instead, we detect the presence of the important flow features and adaptively refine the grid around the flow features.

In this work, we extend the single-grid algorithm of [16] to an adaptive hierarchy of grids; we neglect capillary pressure effects and the Ergun equation that were used in [16]. The basic algorithm uses a total-velocity splitting technique to split the governing equations into a hyperbolic saturation equation and an elliptic pressure equation. We use the block-structured adaptive mesh refinement (AMR) approach that was originally developed by Berger and Oliger [8]. In block-structured refinement, the problem is solved on a hierarchy of nested grids – finer grids are embedded within coarser grids. An error estimation procedure is used to determine where fine grids are needed. In addition, we use smaller time steps on the finer grids – this effectively allows us to refine in time, in addition to refining in space.

There are several ways to organize the elliptic solvers to deal with refinement in time. For example, Hornung and Trangenstein [14] use composite solves on the entire hierarchy of grids at every finest-level time step. Our approach is similar in spirit to Almgren et al. [2]. We use two types of elliptic solvers: 1) a composite solver which solves on the hierarchy of grids and 2) a level solver which only solves on a single level of grids. We perform composite solves only at the times when data is defined at all of the levels. At other instances, such as at a fine-grid time step in between two coarse-grid time steps, we use a level solve and a lagged correction of the boundary conditions. This dual solver method is more computationally efficient than doing only composite solves.

Subcycling in time and using a velocity field that results from an elliptic solve implies that we may not have freestream preservation at coarse/fine interfaces. To deal with these freestream problems, we use a volume-discrepancy method based on the work of Acs et al. [1] and Trangenstein and Bell [17]. We introduce an auxiliary advected quantity that measures the amount of the freestream violation and use it to generate an increment in velocity that drives the solution toward freestream preservation.

The rest of this paper describes the extension of [16] to an adaptive hierarchy of grids. Section 2 gives a brief summary of the governing equations, recasts them in a sequential algorithm, and then describes our algorithm for solving them on a single grid. Section 3 describes the modifications to the single-grid algorithm to extend it to a hierarchy of grids. Section 4 shows that the adaptive algorithm uses a smaller number of grid cells than the single-grid algorithm, but still manages to produce the same results.

2. SINGLE-GRID ALGORITHM

In this section, we specify the governing equations, auxiliary relations and boundary conditions for a simplified model of flow in a porous medium. Then we derive a system of equations that is suitable for a sequential solution method and give an overview of the numerical algorithm for a single grid; a more detailed description of the single-grid algorithm can be found in [16]. We assume a basic knowledge of porous media; for a more detailed examination of porous media, see the books by Bear [4] or Collins [12].

2.1. Governing Equations

Our simplified model assumes that the porosity is not time-dependent and the phase densities and viscosities are constant. In addition, we assume that the system contains two components – component A which exists only in the liquid phase and component B which exists only in the gas phase; as a result, we can use the terms "phase" and "component" interchangeably in this paper. We will denote the liquid phase by the subscript L and the gas phase by the subscript G.

With these assumptions, the three main equations governing the flow are conservation of volume, conservation of mass, and Darcy's law:

$$s_L + s_G = 1 \tag{1}$$

$$\phi \frac{\partial(s_p)}{\partial t} + \nabla \cdot \mathbf{v}_p = 0 \tag{2}$$

$$\mathbf{v}_p = -\lambda_p(\nabla P + \gamma_p \nabla z) \tag{3}$$

where $\gamma_p = \frac{\rho_p g}{g_c}$ is a grouping of gravity terms and s_p, ρ_p, $\mathbf{v}_p$, and λ_p represent the saturation, density, velocity and mobility of phase p. In addition, P is the pressure, g is the acceleration due to gravity, ϕ is the porosity, g_c is the gravity conversion factor ($g_c = 1$ in the metric system), t denotes time, and z is the upward-directed coordinate. These three equations are augmented by auxiliary correlations to make the system solvable.

The phase mobility of phase p is defined as:

$$\lambda_p = k \frac{k_{Rp}}{\mu_p} \tag{4}$$

where k is the permeability, k_{Rp} is the relative permeability of phase p, and μ_p is the viscosity of phase p. The expressions for permeability, relative permeability, and phase viscosity are typically problem dependent. In this work, we treat the permeability as a function of porosity (using the Kozeny-Carman equation) and the relative permeability as a function of porosity and liquid saturation. The specific correlations can be found in [16].

For boundary conditions, we specify that there is an inlet at the top of the domain, an outlet at the bottom of the domain, and the other two sides are impermeable walls (these boundary conditions are meant to model a trickle bed reactor). At the impermeable walls, we specify that the normal components of the velocities are zero. At the outlet, we specify the pressure and that the normal derivative of liquid saturation be zero; in effect, this means that we will have no boundary layer at the outlet. At the inlet, we specify the liquid saturation and the gradient of the pressure.

2.2. Total Velocity Formulation

The equations of flow in a porous medium exhibit both elliptic and hyperbolic behavior. For example, pressure effects are instantaneously felt throughout the reservoir, while saturation fronts move at a finite speed [6]. Our numerical algorithm treats these effects separately by splitting the system of governing equations into an elliptic pressure equation and a hyperbolic saturation equation.

In a manner similar to the work of Watts [18], we define a total velocity:

$$\mathbf{v}_T = \mathbf{v}_L + \mathbf{v}_G. \tag{5}$$

As a result of the conservation of volume (1) and conservation of mass (2) equations, the total velocity is divergence-free. Using the definition of total velocity, we obtain a pressure equation and a conservation of mass equation:

$$\nabla \cdot [(\lambda_L + \lambda_G)(\nabla P)] = \nabla \cdot [(\lambda_G \gamma_G + \lambda_L \gamma_L)(\nabla z)] \tag{6}$$

$$\phi \frac{\partial s_L}{\partial t} + \nabla \cdot \mathbf{F}(s_L, \mathbf{v}_T) = 0 \tag{7}$$

where

$$\begin{aligned} \mathbf{F}(s_L, \mathbf{v}_T) &= \frac{\lambda_L(\mathbf{v}_T - \mathbf{G}\,\lambda_G)}{\lambda_L + \lambda_G} \\ \mathbf{G} &= (\gamma_L - \gamma_G)\nabla z. \end{aligned}$$

In addition, we can write the total velocity as a function of pressure and phase mobilities:

$$\mathbf{v}_T = -(\lambda_L + \lambda_G)\nabla P - (\lambda_L \rho_L + \lambda_G \rho_G) g \nabla z. \tag{8}$$

2.3. Numerical Algorithm

We cover the domain with a mesh of finite-volume grid cells. We use one of two different two-dimensional coordinate systems for these grid cells – 1) a Cartesian coordinate system (x-z) and 2) a cylindrical coordinate system (r-z). In the Cartesian coordinate system, the grid cells are rectangular of size $\triangle x$ by $\triangle z$, and are indexed in the x-direction by i and in the z-direction by j. In the cylindrical coordinate system, the grid cells are of size $\triangle r$ by $\triangle z$, and are indexed in the r-direction by i and in the z-direction by j. We discretize in time using the index n, such that the time step $\triangle t$ is the difference between discrete times t^n and t^{n+1}. Saturations and pressures are defined at cell centers, such that $s^n_{i,j} \approx s((i+0.5)\triangle x, (j+0.5)\triangle z, t^n)$. Phase mobilities and the normal components of velocities are defined at cell edges and use half indices, such that $v^n_{i+\frac{1}{2},j}$ represents the x-velocity at the "right" edge of cell (i, j).

The sequential algorithm advances the liquid saturation from time t^n to time t^{n+1} by solving a sequence of hyperbolic and elliptic equations. The starting point for our approach is the multidimensional upwind method in [11], extended in [5] to combined hyperbolic and elliptic equations. We follow the treatment of [16], particularly in the treatment of the nonlinear hyperbolic terms.

We denote variables at the current time by the superscript n and variables at the new time by the superscript $n+1$. In addition, we denote temporary predicted variables at time $n+1$ by the superscript $\tilde{}$. With this notation, we can express the predictor-corrector scheme to advance the saturation from n to $n+1$ as:

1. Compute the pressure and velocity at the current time step.

We compute the pressure at the current time, P^n, by solving the pressure equation:

$$-\nabla \cdot [(\lambda_G^n + \lambda_L^n)(\nabla P^n)] = -\nabla \cdot [(\lambda_G^n \gamma_G + \lambda_L^n \gamma_L)(\nabla z)].$$

Next, we compute the total velocity at the current time, $\mathbf{v}_T^n$:

$$\mathbf{v}_T^n = -(\lambda_L^n + \lambda_G^n)\nabla P^n - (\lambda_L^n \gamma_L + \lambda_G^n \gamma_G)\nabla z.$$

2. Trace the liquid saturation to cell edges at the half time step.

We use the velocity and saturation at the current time in Godunov's method to compute liquid saturation at the half time step at cell edges, $s_{EDGE}^{n+\frac{1}{2}}$.

3. Approximate the velocity at the half time step.

We compute the velocity at the half time step by averaging the velocity at the current time step, $\mathbf{v}_T^n$, with the predicted velocity at the next time step, $\tilde{\mathbf{v}}_T$. The first step in computing $\tilde{\mathbf{v}}_T$ is to predict the saturation at the next time step, $\tilde{s}_L$:

$$\phi \frac{\tilde{s}_L - s_L^n}{\triangle t} = -\nabla \cdot \mathbf{F}(s_{EDGE}^{n+\frac{1}{2}}, \mathbf{v}_T^n).$$

where $\mathbf{F}$ is an approximation to the flux at cell edges. We use this predicted saturation to compute the predicted pressure at the next time step, $\tilde{P}$:

$$-\nabla \cdot [(\tilde{\lambda}_G + \tilde{\lambda}_L)(\nabla \tilde{P})] = -\nabla \cdot [(\tilde{\lambda}_G \gamma_G + \tilde{\lambda}_L \gamma_L)(\nabla z)].$$

Again, the $\tilde{}$ superscript on the phase mobilities indicates that they are computed using the predicted saturation $\tilde{s}$. Next, we predict the velocity at the next time step, $\tilde{\mathbf{v}}_T$:

$$\tilde{\mathbf{v}}_T = -(\tilde{\lambda}_L + \tilde{\lambda}_G)\nabla \tilde{P} - (\tilde{\lambda}_L \gamma_L + \tilde{\lambda}_G \gamma_G)\nabla z.$$

Finally, we average the predicted velocity and the current velocity to obtain the velocity at the half time step, $\mathbf{v}_T^{n+\frac{1}{2}}$:

$$\mathbf{v}_T^{n+\frac{1}{2}} = \frac{1}{2}(\mathbf{v}_T^n + \tilde{\mathbf{v}}_T).$$

4. Compute the liquid saturation at the next time step.

Now that we have both a saturation and a velocity at the half time step, we can compute a second-order accurate saturation at the next time step, s_L^{n+1}:

$$\phi \frac{s_L^{n+1} - s_L^n}{\triangle t} = -\nabla \cdot \mathbf{F}(s_{EDGE}^{n+\frac{1}{2}}, \mathbf{v}_T^{n+\frac{1}{2}}).$$

The use of Godunov's method and methods for solving the saturation equation and pressure equation are described in detail in [16].

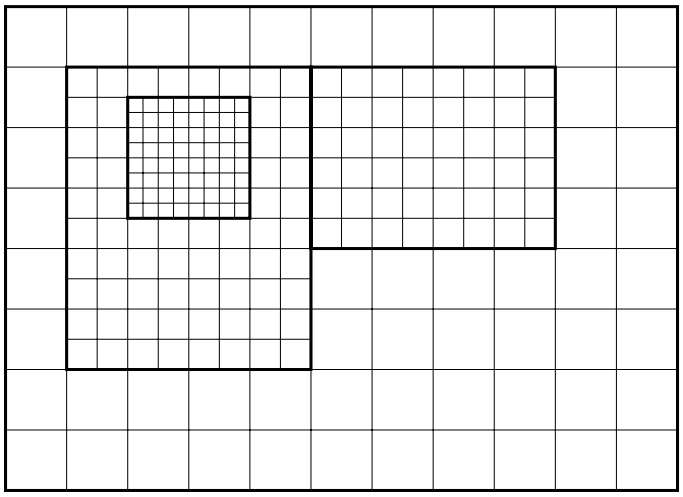


FIG. 1. Example of a grid hierarchy with two levels of refinement. The refined levels are nested inside coarser levels.

3. ADAPTIVE ALGORITHM

In this work, we use the block-structured adaptive mesh refinement approach that was originally developed by Berger and Oliger [8]. In block-structured refinement, the problem is solved on a hierarchy of nested grids. This section begins with a description of the AMR grid hierarchy and then focuses on the differences that arise due to the presence of a hierarchy of grids, such as synchronization of data between levels and freestream preservation. Finally, it gives a brief overview of the adaptive algorithm on a hierarchy of grids and discusses several different algorithms for determining which cells need refinement.

3.1. AMR Grid Hierarchy

In a block-structured approach to adaptive mesh refinement, the problem is solved on a hierarchy of levels; each level consists of a collection of rectangular grids. We index these levels by the superscript ℓ, where $\ell = 0$ is the coarsest level and $\ell = \ell_{max}$ is the finest level. We denote the problem domain covered by level ℓ grids by Ω^ℓ. We require that Ω^0 cover the entire problem domain, while finer levels generally only cover part of the problem domain. Figure 1 shows an example of a grid hierarchy that contains two levels of refinement.

We use the superscript * to denote cell edges; therefore, the union of cell edges on level ℓ is denoted by $\Omega^{\ell,*}$. We define $\partial\Omega^{\ell,*}$ as the boundary of grids on level ℓ; we note that $\partial\Omega^{\ell,*}$ is a subset of $\Omega^{\ell,*}$. Finally, we define a projection operator, $\mathcal{P}(\Omega^{\ell+1})$ as the set of level ℓ cells that are covered by level $\ell+1$ cells. The projection operator also applies to cell edges; for example, $\mathcal{P}^(\Omega^{\ell+1,*})$ are the edges of cells on level ℓ that are covered by edges of cells on level $\ell+1$ grids.

Each level has its own mesh spacing; the mesh spacing for level ℓ in the x-direction is denoted by $\triangle x^\ell$, while the mesh spacing in the z-direction is denoted by $\triangle z^\ell$. We define the refinement ratio for level ℓ, n^ℓ_{ref}, to be the ratio of mesh spacings

between level ℓ and level $\ell+1$:

$$n_{ref}^{\ell} \equiv \frac{\triangle x^{\ell}}{\triangle x^{\ell+1}} \equiv \frac{\triangle z^{\ell}}{\triangle z^{\ell+1}}.$$

In general, there is no restriction on the value of the refinement ratio except that it be greater than one; however, in this work, we use refinement ratios that are powers of two in order to facilitate convergence of the multigrid-based elliptic solvers. We do not require each level to have the same refinement ratio.

We require that the grids satisfy two proper nesting conditions. The first condition is that if a cell is refined, it is completely refined; there is no partial cell refinement. The second condition is that $\partial\Omega^{\ell}$ either resides in the interior of $\Omega^{\ell-1}$ or on a physical boundary. These conditions substantially simplify the communication of data between levels and the imposition of boundary conditions.

3.1.1. Variables and Operators

In addition to making the distinction between cell-centered and edge-centered variables, we also classify variables as either level or composite variables. Level variables are valid in all of the grid cells on a level and are computed without knowledge of data from finer or coarser levels. Composite variables, denoted by the superscript *comp*, are only valid in grid cells that are not covered by finer grid cells. Thus, a cell-centered composite variable on level ℓ only has valid data on $\Omega^{\ell} - \mathcal{P}(\Omega^{\ell+1})$. Edge-centered composite variables are defined in a similar manner; an edge-centered composite variable on level ℓ is only valid on $\Omega^{\ell,*} - \mathcal{P}(\Omega^{\ell+1,*})$.

The operators in the adaptive algorithm require that all grid cells have data, so in cells that are covered by finer grid cells, we average down the finer grid data to the coarse grid cells. We define the averaging operator, $<>$, as the operator that does simple arithmetic averaging to transfer data from level $\ell+1$ to level ℓ. For cell-centered variables, we average the values of the $(n_{ref}^{\ell})^2$ overlying level $\ell+1$ cells. For edge-centered variables, we average the values of the n_{ref}^{ℓ} overlying level $\ell+1$ cell edges.

A vector field is a special type of edge-centered variable – it represents a vector of data where each component of the field is only defined on one set of edges of the cell. The x-component of the vector field is only defined on the x-edges of the cell, while the z-component of the vector field is only defined on the z-edges of the cell. Vector fields may be either composite or level variables.

Gradients.

The generic gradient operator, G, takes a cell-centered scalar and maps it into an edge-centered vector field. If we consider a cell-centered scalar ϕ, we define the x-component and z-component of the gradient field as:

$$G(\phi)|_{i+\frac{1}{2},j} = \frac{\phi_{i+1,j} - \phi_{i,j}}{\triangle x}, \qquad G(\phi)|_{i,j+\frac{1}{2}} = \frac{\phi_{i,j+1} - \phi_{i,j}}{\triangle z}. \tag{9}$$

We define the level gradient operator acting on level ℓ, $G^{\ell}(\phi^{\ell}, \phi^{\ell-1})$, by Eq. (9). If one of the grid cells necessary to compute the gradient falls outside the boundaries of the grid, we classify the boundary as one of three different types: 1) a physical boundary, 2) a boundary with another grid at the same level of refinement, and 3)

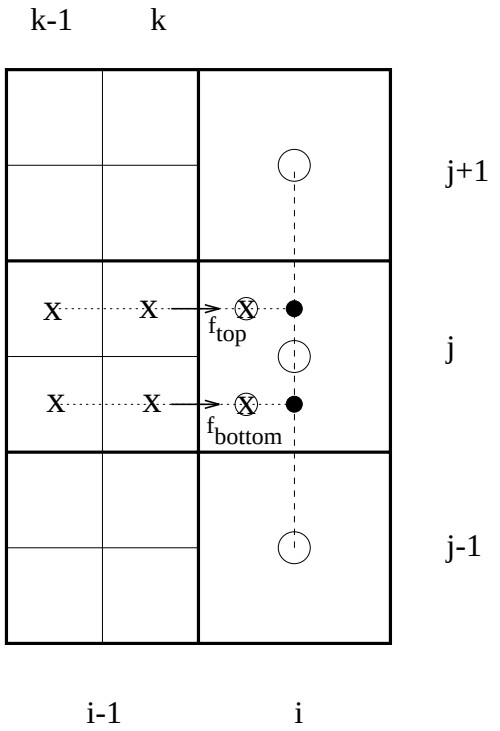


FIG. 2. The interpolation stencil and computation of fluxes at a coarse/fine interface. A polynomial is first fit through the three coarse cells (the empty circles); we use this polynomial to compute the intermediate values (the filled circles). Then, a polynomial is fit through the two fine cells (the X's) and the intermediate values (the filled circles); we use this polynomial to compute the values in the ghost cells (the circles with X's).

a boundary with another grid at a coarser level of refinement. If a grid boundary is a physical boundary, then the cell is filled by enforcing the physical boundary condition. For a boundary with another grid at the same level of refinement, we copy the data from the other grid. If the boundary of a grid is not with another grid at the same level or a physical boundary, then the grid cell is filled using data from coarser grids; as a result of the proper nesting condition, the grid cell values can be determined using only data from the next coarser level. As shown in Figure 2, a polynomial is first fit through the three coarse cells (the empty circles) adjacent to the interface and is used to compute the intermediate values (the filled circles). Then, a polynomial is fit through two fine cells (the X's) and the newly interpolated coarse cells (the filled circles); this polynomial is used to compute the values in the ghost cells (the circles with X's). If the three coarse grid cells do not exist, then we modify the coarse grid polynomial by shifting it one coarse grid cell.

The composite gradient operator acting on level ℓ, $G^{comp,\ell}(\phi^\ell, \phi^{\ell-1})$, is also defined by Eq. (9) and it fills cells outside of the grid in the same manner as the level gradient operator. However, the composite gradient operator is only valid at cell edges that are not covered by edges of finer cells, i.e. $\Omega^{\ell,*} - \mathcal{P}(\Omega^{\ell+1,*})$.

Divergence and Reflux Divergence.

The generic divergence operator, D, takes an edge-centered vector field and maps it into a cell-centered scalar. We define the divergence operator acting on an edge-

centered vector $\mathbf{f}$ as:

$$D(\mathbf{f})|_{i,j} = \frac{f^x_{i+\frac{1}{2},j} - f^x_{i-\frac{1}{2},j}}{\triangle x} + \frac{f^z_{i,j+\frac{1}{2}} - f^z_{i,j-\frac{1}{2}}}{\triangle z}. \tag{10}$$

We define a level divergence operator acting on level ℓ, $D^\ell(\mathbf{f}^\ell)$ by Eq. (10). Since the vector field contains all of the necessary edge-centered values to compute the operator, there is no need to use boundary conditions to fill ghost cells.

The composite divergence operator on level ℓ, $D^\ell(\mathbf{f}^\ell, \mathbf{f}^{\ell+1})$, is also defined by Eq. (10); however, we alter the composite operator at interfaces with finer grids, i.e. on $\mathcal{P}(\partial\Omega^{\ell+1,*})$. At this interface, we have an edge-centered, coarse-grid value and n^ℓ_{ref} edge-centered, fine-grid values with no guarantee that they will be equivalent. To force them to be equivalent, we assume that the fine-grid values are more accurate and replace the coarse-grid value with the average of the fine-grid values. For example, in Figure 2 we compute the divergence on level ℓ as:

$$D(\mathbf{f})|_{i,j} = \frac{f^x_{i+\frac{1}{2},j} - < f^{x,\ell+1} >_{i-\frac{1}{2},j}}{\triangle x} + \frac{f^z_{i,j+\frac{1}{2}} - f^z_{i,j-\frac{1}{2}}}{\triangle z}$$

where in this example

$$< f^{x,\ell+1} >_{i-\frac{1}{2},j} = \frac{f_{top} + f_{bottom}}{2}.$$

We also define a special type of edge-centered vector field called a flux register. We denote the flux register for a variable f on the level $\ell/(\ell+1)$ interface by $\delta F^{\ell+1,f}$ (notationally, flux registers are associated with the finer level). The flux register is defined as:

$$\delta F^{\ell+1,f} = < \mathbf{f}^{\ell+1} \cdot \hat{n} > - \mathbf{f}^\ell \cdot \hat{n} \quad \text{on} \quad \mathcal{P}(\partial\Omega^{\ell+1,*}) \tag{11}$$

where $\hat{n}$ is the outward normal vector for the coarse/fine interface. A flux register on level ℓ is only valid on $\mathcal{P}(\partial\Omega^{\ell+1,*})$.

Finally, we define a reflux divergence operator, $D_R(\delta F)$, which takes edge-centered flux registers and computes cell-centered scalars:

$$D_R(\delta F^{\ell+1,f})_{i,j} = \sum_{e \in \mathcal{P}(\partial\Omega^{\ell+1,*})} \frac{\pm A_e}{VOL_{i,j}} \, \delta \mathrm{F}^{\ell+1,\mathrm{f}} \tag{12}$$

where A_e is the cross-sectional area of the edge and $VOL_{i,j}$ is the volume of cell (i,j). It is implied that the edges e in the summation are only those edges which are adjacent to cell (i,j). The $+$ or $-$ sign is chosen based on which side of the coarse cell the flux register is defined; we use $+$ for the high side of a coarse cell and $-$ for the low side of a coarse cell. With these definitions, we note that we can compute the composite divergence using flux registers and the level divergence:

$$D^{comp,\ell}(\mathbf{f}^\ell, \mathbf{f}^{\ell+1}) = D^\ell(\mathbf{f}^\ell) + D_R(\delta F^{\ell+1,f}) \quad \text{on} \quad \Omega^\ell - \mathcal{P}(\Omega^{\ell+1}).$$

Elliptic Operator.
We define the elliptic operator L as:

$$L(\phi) = \nabla \cdot \beta \nabla \phi \tag{13}$$

where β is an edge-centered vector field and ϕ is a cell-centered variable. We can write L in terms of the generic divergence and gradient operators:

$$L(\phi) = D(\beta G(\phi)). \tag{14}$$

Discretizing these generic operators, we obtain:

$$\begin{aligned} L(\phi_{i,j}) &= \frac{\beta_{i+\frac{1}{2},j}\left(\phi_{i+1,j} - \phi_{i,j}\right) - \beta_{i-\frac{1}{2},j}\left(\phi_{i,j} - \phi_{i-1,j}\right)}{\triangle x^2} \\ &+ \frac{\beta_{i,j+\frac{1}{2}}\left(\phi_{i,j+1} - \phi_{i,j}\right) - \beta_{i,j-\frac{1}{2}}\left(\phi_{i,j} - \phi_{i,j-1}\right)}{\triangle z^2}. \end{aligned} \tag{15}$$

We define two types of elliptic operator: 1) a composite elliptic operator and 2) a level elliptic operator. The composite elliptic operator, $L^{comp,\ell}(\phi^\ell)$, discretizes Eq. (14) using composite variables and composite operators and acts on the entire hierarchy of levels. This operator uses data from the next coarser and next finer levels. In cells that are away from the Ω^ℓ boundary, the operator $L^{comp,\ell}(\phi^\ell)$ is the normal elliptic operator (15). In cells on the $\Omega^\ell/\Omega^{\ell-1}$ boundary (i.e. next to a coarser grid), we use quadratic interpolation to place values in the border cells, and then evaluate the operator as usual. In cells that border the $\Omega^\ell/\Omega^{\ell+1}$ boundary (i.e. next to a finer grid), we use the flux matching conditions to evaluate fluxes at the boundary (see Section 3.2.3).

On the other hand, a level elliptic operator discretizes Eq. (14) using level variables and level operators and acts on a single level; this operator knows about the next coarser level, but it has no knowledge of the next finer level. As a result, $L^\ell(\phi^\ell, \phi^{\ell-1})$ is the same as $L^{comp,\ell}(\phi^\ell)$ except that it does not use the coarse/fine flux matching condition at boundaries with fine grids.

3.2. AMR for Porous Media Equations

In Section 2, we discussed the solution of the porous media equations on a single grid. In this subsection, we describe the implementation of the algorithm on a hierarchy of grids, focusing on the differences from the single-grid algorithm; these differences include the selection of the time step and grid synchronization issues, such as multilevel matching conditions and freestream preservation. Finally, we state our algorithm for solving the coupled system of equations.

3.2.1. Time Stepping

In this work, we subcycle in time – we advance finer grids at a time step smaller than coarse grids; as a result, we only solve on part of the grid hierarchy at each time step. We use the CFL condition to choose the time step on the coarsest level. Denoting the time step on level ℓ by $\triangle t^\ell$, we choose the time step on finer levels

```
timeStep(t, ℓ)
    advance(ℓ, t, △t^ℓ)
    if (ℓ < ℓ_max)
      for (i = 0, n^ℓ_ref − 1)
          timeStep(t + i ∗ △t^{ℓ+1}, ℓ + 1)
      end for
      Synchronize(ℓ, ℓ + 1)
    end if
end timeStep
```

FIG. 3. Example of subcycling

using:

$$\triangle t^{\ell+1} = \triangle t^{\ell} \frac{1}{n^{\ell}_{ref}}.$$

Subcycling in time can be thought of as a three-step recursive process on level ℓ (see Figure 3). The first step is to advance level ℓ in time using boundary conditions from level $\ell - 1$ and from physical boundaries. The second step is to advance level $\ell + 1$, if it exists, n^{ℓ}_{ref} times using a time step of $\frac{\triangle t^{\ell}}{n^{\ell}_{ref}}$. The third and final step is to synchronize the data and the fluxes between coarse and fine levels.

3.2.2. Coarse/Fine Data Matching

The first synchronization issue is coarse/fine data matching. In this case, the saturation in a coarse-grid cell may not be equivalent to the saturations in the fine-grid cells that cover it. This mismatch is simple to correct – we average down the saturation from the fine level to the coarse level using the averaging operator <>.

3.2.3. Coarse/Fine Flux Matching

The next synchronization issue is coarse/fine flux matching. At a coarse/fine interface, there is no reason to expect that the coarse-grid fluxes will equal the fine-grid fluxes. For example, consider the coarse/fine interface shown in Figure 2. There is no guarantee that the two fine-grid fluxes (f_{bottom} and f_{top}) are equivalent to the coarse-grid flux (f_{coarse}). This flux inconsistency appears in both the elliptic and hyperbolic equations.

Hyperbolic Equation.

For hyperbolic problems, the coarse/fine flux matching condition implies that the time-averaged sum of the fine-grid fluxes should be equal to the coarse-grid flux. To enforce this condition, we use the correction scheme of Berger and Colella [7]. This scheme uses flux registers to store the difference in fluxes between levels at coarse/fine interfaces. After advancing the fine level, the flux registers are used to modify the coarse-grid fluxes.

Consider refluxing in the x-direction for cell (i, j) in Figure 2. First, we advance the solution one time step on the coarse grid and initialize the flux register, δF, with the coarse-grid flux. Then, we advance the solution n^{ℓ}_{ref} times on the fine grid. At the end of each fine time step, we increment the flux register with the sum

of the fine-grid fluxes at the coarse/fine interface. After n^{ℓ}_{ref} fine time steps, the coarse and fine grids are at the same time and the flux register contains:

$$\delta F_{i-\frac{1}{2},j} = -\triangle z^C \triangle t^C F^C_{i-\frac{1}{2},j} + \triangle z^F \triangle t^F \sum_{p=0}^{p=n^{\ell}_{ref}-1} F^F_{k+\frac{1}{2},m+p}.$$

We reflux the coarse-grid solution as:

$$\begin{aligned} s_{i,j} &= s_{i,j} + \frac{D_R(\delta F)}{\phi_{i,j}} \\ &= s_{i,j} - \frac{\delta F_{i-\frac{1}{2},j}}{\phi_{i,j} VOL_{i,j}} \end{aligned}$$

where D_R is the reflux divergence operator (12). The minus sign was chosen because we are dealing with the lower edge of the cell. The cross-sectional area term that normally appears in the definition of the reflux divergence operator was included in the computation of the flux registers, while the $\frac{1}{\phi}$ term accounts for the fact that the volume used in the flux register computation is total volume, not pore volume.

Elliptic Equation.

The obvious approach to solving an elliptic equation on a grid hierarchy is to solve the problem on the coarse grid and then use the coarse-grid solution as boundary conditions when solving on the fine grid. However, this approach does not produce the correct composite solution – we lose accuracy at the coarse/fine interface because the solution at the coarse level never sees the effects of the solution at the fine level.

To fix this problem, we rewrite the discrete elliptic operator (15) in flux differencing form:

$$L(\phi) = \frac{F^X_{i+\frac{1}{2},j} - F^X_{i-\frac{1}{2},j}}{\triangle x} + \frac{F^Z_{i,j+\frac{1}{2}} - F^Z_{i,j-\frac{1}{2}}}{\triangle z}$$

where the discrete edge-centered flux in the x-direction, $F^X_{i+\frac{1}{2},j}$, is:

$$F^X_{i+\frac{1}{2},j} = \beta_{i+\frac{1}{2},j}\left(\frac{\phi_{i+1,j} - \phi_{i,j}}{\triangle x}\right)$$

and z-direction fluxes are defined analogously. To enforce the flux matching condition we replace the coarse-grid flux with the sum of the fine-grid fluxes. This matching condition is built into the elliptic operator.

3.2.4. Time-Dependent Elliptic Boundary Conditions

Subcycling in time creates time-centering problems with boundary conditions for the elliptic pressure equation. Consider a two-level problem with a refinement ratio of two. We advance level 0 from time $t^{0,n}$ to $t^{0,n+1}$. Then, we attempt to advance level 1 from $t^{1,2n}$ to $t^{1,2n+2}$; however, when we attempt to solve at $t^{1,2n+1}$ $= t^{0,n+\frac{1}{2}}$, there are no cells on level 0 at the proper time to use for boundary conditions. Therefore, we need to interpolate in time on level 0 between t^n and

t^{n+1} to obtain a pressure with the correct time-centering; then, we will need to interpolate in space to obtain the pressure at the correct spatial position.

However, this solution method produces a flux mismatch – the coarse grid never sees the results of the fine-grid computation. As a result, we obtain the same accuracy as if we had only solved on the coarse level. Hornung and Trangenstein [14] dealt with this matching condition by using a multilevel solver at every finest-level time step and interpolating coarse-grid saturations as necessary; however, this method is computationally expensive. In this work, we use a different approach – we combine the time interpolation of coarse-grid data with a lagged correction scheme. We synchronize pressure at the beginning of each time step on each level (performing a composite solve, if necessary); then, we compute a pressure correction – the difference between the composite pressure and the level pressure – and use it to account for the presence of other levels in the AMR hierarchy. As a result, we can use level solvers and include the pressure correction to account for other levels. This algorithm does require us to lag the pressure correction for a time step; however, we have found that the error introduced by lagging the pressure correction is much smaller than the discretization error.

3.2.5. Freestream Preservation

Freestream preservation refers to the idea that in a uniform flow field, constant properties (scalar fields, velocities, etc.) should remain constant. In this work, freestream preservation may be violated at coarse/fine interfaces when we reflux. We can illustrate this violation by considering a hyperbolic conservation law of the form:

$$\frac{\partial s}{\partial t} + \nabla \cdot (\mathbf{F}(s, \mathbf{v}_T)) = 0 \tag{16}$$

where

$$\mathbf{F}(s, \mathbf{v}_T) = s\mathbf{v}_T.$$

Let a coarse/fine interface occur in the midst of a region of constant saturation, s_0. In that case, the flux register on level ℓ can be written as:

$$\begin{aligned} \delta F^\ell &= -s^\ell \mathbf{v}_T^\ell \cdot \hat{n} + \frac{1}{n_{ref}^\ell} \sum < s^{\ell+1} \mathbf{v}_T^{\ell+1} \cdot \hat{n} > \\ &= s_0 \left(\mathbf{v}_\mathrm{T}^\ell \cdot \hat{\mathrm{n}} - \frac{1}{\mathrm{n}_\mathrm{ref}^\ell} \sum < \mathbf{v}_\mathrm{T}^{\ell+1} \cdot \hat{\mathrm{n}} > \right) \end{aligned}$$

where the summation is over the fine time steps and $\hat{n}$ is the normal vector. For the saturation to remain constant in the region, $\delta F^\ell = 0$; otherwise, refluxing will cause the saturation to change. This requirement simplifies to a constraint on the velocity – the coarse velocity must equal the time averaged sum of the fine velocities. However, since the velocities are dependent on different elliptic pressure solves, there is no reason to expect this constraint to be satisfied; as a result, we expect our algorithm to violate freestream preservation.

Almgren et al. [2] used a multilevel correction scheme that involved both elliptic solves and interpolation of hyperbolic corrections onto finer grids. With this

scheme, they were able to implement an algorithm that was exactly freestream preserving. We use a different correction method that is only approximately freestream preserving, but is less complicated to implement. Our correction method is based on the volume-discrepancy formulation used by Acs et al. [1] and Trangenstein and Bell [17]. In their work, the volume balance is linearized and hence, fluid volume is not conserved. At the end of the each time step, they compute the volume discrepancy; this volume discrepancy is then used as a correction term in their pressure equation.

To implement our freestream correction, we define an additional freestream variable, V. We initially set $V = 1$ and then we advance it as a conserved quantity:

$$\frac{\partial V}{\partial t} + \nabla \cdot (\mathbf{v}_T V) = 0. \tag{17}$$

Since the freestream quantity should remain constant, the extent that $V \neq 1$ measures the accumulated compression or expansion of the fluid due to the coarse/fine boundary conditions. We note that (17) is solved with the same algorithm as the liquid saturation equation – the only difference is that the flux function for the freestream quantity, $\mathbf{F}^V(\mathbf{v}_T, V)$, is different.

We use the freestream quantity in a correction term. Performing a Taylor series expansion on $V(x(t), t + \triangle t)$ and setting $V(x(t), t + \triangle t) = 1$ (since we want to predict the change necessary to set V equal to one at the new time), we obtain:

$$\nabla \cdot \mathbf{v}_T = \frac{V - 1}{V \triangle t}.$$

Recall that our pressure equation enforces the constraint $\nabla \cdot \mathbf{v}_T = 0$. We modify the right-hand side of the equation for pressure by adding the $\frac{V-1}{V\triangle t}$ term. To ensure that we do not induce an instability, we scale the freestream correction term by η, where $\eta < 1$. In this work, we use $\eta = 0.90$. As a result, the pressure equation, with the freestream correction, is:

$$\nabla \cdot [(\lambda_L + \lambda_G)(\nabla P)] = \nabla \cdot [(\lambda_G \gamma_G + \lambda_L \gamma_L)(\nabla z)] + \eta \frac{V - 1}{V \triangle t}.$$

3.2.6. Algorithm

The adaptive algorithm follows the same general predictor-corrector scheme as the single-grid algorithm, but there are several differences between the algorithms due to synchronization issues; these differences were discussed in the previous subsections. The algorithm to advance the solution on a level can be divided into six steps; the steps are numbered 0-5 so that they correspond to the steps in the single-grid algorithm in Section 2.3. In outline form, the algorithm to advance the solution on level ℓ from n to $n + 1$ is:

0. Compute the composite pressure and composite velocity at the current time.

If we are doing the first time step at the current time, then we need to compute the composite pressure, $p^{comp,n}$ and the composite velocity, $\mathbf{v}^{comp,n}$ on levels ℓ to ℓ_{max}. If we are not doing the first time step at the current time, these quantities have already been computed during the update of a coarser time level.

First, we compute the composite pressure by solving the pressure equation modified with the freestream preservation term:

$$-\nabla \cdot (\lambda_L^{n,\ell} + \lambda_G^{n,\ell}) \nabla p^{comp,n} = -\nabla \cdot [(\lambda_G^{n,\ell} \gamma_G + \lambda_L^{n,\ell} \gamma_L)(\nabla z)] + \frac{1}{\triangle t^\ell} \frac{V^{n,\ell} - 1}{V^{n,\ell}}.$$

This equation is solved using a composite elliptic solver. If we are on the coarsest level, then we only need to use physical boundary conditions; however, if we are not on the coarsest level, then we use a combination of physical boundary conditions and boundary conditions from level $\ell - 1$. The level $\ell - 1$ conditions are interpolated linearly in time between $p^{\ell-1}$ and $\tilde{p}^{\ell-1}$. Since both $p^{\ell-1}$ and $\tilde{p}^{\ell-1}$ are computed from level solves, we need to add in the pressure correction $e^{\ell-1}$ (defined in Step 1) that accounts for the presence of other levels. In compact notation, this boundary condition is stated as:

$$p^{comp,\ell} = I_{TX}(p^{\ell-1}, \tilde{p}^{\ell-1}) + e^{\ell-1} \quad \text{on} \quad \partial\Omega^\ell$$

where I_{TX} indicates interpolation in both space and time.

Next, we compute the composite total velocity, $\mathbf{v}_T^{comp,n,\ell}$ on levels ℓ_{max} to ℓ. We compute the velocity as a function of the composite pressure; however, when the a grid on the current level is covered by a finer grid, we average down the velocity from the finer grid:

$$\mathbf{v}_T^{comp,n,\ell} = \begin{cases} -(\lambda_L^{n,\ell} + \lambda_G^{n,\ell}) \nabla p^{comp,n,\ell} - (\lambda_L^{n,\ell} \gamma_L + \lambda_G^{n,\ell} \gamma_G) \nabla z & \text{on } \Omega^{\ell,*} - \mathcal{P}(\Omega^{\ell+1,*}) \\ < \mathbf{v}_T^{comp,n,\ell+1} > & \text{on } \mathcal{P}(\Omega^{\ell+1,*}). \end{cases}$$

1. Compute the level pressure and level velocity at the current time step.

We compute the level pressure, $p^{n,\ell}$:

$$-\nabla \cdot (\lambda_L^{n,\ell} + \lambda_G^{n,\ell}) \nabla p^{n,\ell} = -\nabla \cdot [(\lambda_G^{n,\ell} \gamma_G + \lambda_L^{n,\ell} \gamma_L)(\nabla z)] + \frac{1}{\triangle t^\ell} \frac{V^{n,\ell} - 1}{V^{n,\ell}}.$$

For boundary conditions, we interpolate in time between $p^{\ell-1}$ and $\tilde{p}^{\ell-1}$. We also compute the level velocity, $\mathbf{v}_T^{n,\ell}$, as a function of the level pressure:

$$\mathbf{v}_T^{n,\ell} = -(\lambda_L^{n,\ell} + \lambda_G^{n,\ell}) \nabla p^{n,\ell} - (\lambda_L^{n,\ell} \gamma_L + \lambda_G^{n,\ell} \gamma_G) \nabla z.$$

Finally, we compute the pressure correction, $e^{n,\ell}$:

$$e^{n,\ell} = p^{comp,n,\ell} - p^{n,\ell}.$$

This corrections account for the presence of other levels in the grid hierarchy.

2. Trace the scalar variables to cell edges at half time step.

We use the composite velocity in Godunov's method to compute the advected quantities at the half time step at cell edges: $s_{EDGE}^{n+\frac{1}{2},\ell}, V_{EDGE}^{n+\frac{1}{2},\ell}$.

3. Approximate the velocity at the half time step.

We compute the velocity at the half time step by averaging the velocity at the current time step, $\mathbf{v}_T^{comp,n,\ell}$, with the predicted velocity at the next time step, $\tilde{\mathbf{v}}_T^\ell$.

The first step in computing $\tilde{\mathbf{v}}_T^\ell$ is to predict the saturation and freestream quantity at the next time step, $\tilde{s}^\ell$ and $\tilde{V}^\ell$:

$$\begin{aligned}\phi\frac{\tilde{s}^\ell - s^{n,\ell}}{\triangle t^\ell} &= -\nabla\cdot\mathbf{F}^s(s^{n+\frac{1}{2},\ell}, \mathbf{v}_T^{comp,n,\ell})\\ \phi\frac{\tilde{V}^\ell - V^{n,\ell}}{\triangle t^\ell} &= -\nabla\cdot\mathbf{F}^V(V^{n+\frac{1}{2},\ell}, \mathbf{v}_T^{comp,n,\ell}).\end{aligned}$$

We perform a level elliptic solve using these predicted quantities to compute the predicted pressure at the next time step, $\tilde{p}^\ell$:

$$-\nabla\cdot(\tilde{\lambda}_L^{n,\ell} + \tilde{\lambda}_G^{n,\ell})\nabla\tilde{p}^\ell = -\nabla\cdot[(\tilde{\lambda}_G^{n,\ell}\gamma_G + \tilde{\lambda}_L^{n,\ell}\gamma_L)(\nabla z)] + \frac{1}{\triangle t^\ell}\frac{\tilde{V}^\ell - 1}{\tilde{V}^\ell}.$$

For the boundary conditions, we interpolate in time between $p^{\ell-1}$ and $\tilde{p}^{\ell-1}$ and add in the pressure correction to make this level solve equivalent to a composite solve. In compact notation, this boundary condition is:

$$\tilde{p}^\ell = I_{TX}(p^{\ell-1}, \tilde{p}^{\ell-1}) + e^\ell \quad \text{on} \quad \partial\Omega^\ell$$

Next, we predict the velocity at the next time step, $\tilde{\mathbf{v}}_T^\ell$, as a function of the predicted pressure:

$$\tilde{\mathbf{v}}_T^\ell = -(\tilde{\lambda}_L^{n,\ell} + \tilde{\lambda}_G^{n,\ell})\nabla\tilde{p}^\ell - (\tilde{\lambda}_L^{n,\ell}\gamma_L + \tilde{\lambda}_G^{n,\ell}\gamma_G)\nabla z$$

Finally, we average the composite velocity and the predicted velocity to obtain the velocity at the half time step, $\mathbf{v}_T^{n+\frac{1}{2},\ell}$:

$$\mathbf{v}_T^{n+\frac{1}{2},\ell} = \frac{1}{2}(\mathbf{v}_T^{n,\ell} + \tilde{\mathbf{v}}_T^\ell).$$

4. Compute the advected quantities at the next time step.

Now that we have both a saturation and a velocity at the half time step, we can compute a second-order accurate saturation and freestream quantity at the next time step:

$$\begin{aligned}\phi\frac{s^{n+1,\ell} - s^{n,\ell}}{\triangle t^\ell} &= -\nabla\cdot\mathbf{F}^s(s_{EDGE}^{n+\frac{1}{2},\ell}, \mathbf{v}_T^{n+\frac{1}{2},\ell})\\ \phi\frac{V^{n+1,\ell} - V^{n,\ell}}{\triangle t^\ell} &= -\nabla\cdot\mathbf{F}^V(V_{EDGE}^{n+\frac{1}{2},\ell}, \mathbf{v}_T^{n+\frac{1}{2},\ell}).\end{aligned}$$

If a finer level exists, we initialize the level $\ell+1$ flux registers, $\delta F^{\ell+1}$, with the level ℓ fluxes (in this step, we are concerned with the level $\ell/(\ell+1)$ interface, so the level ℓ fluxes are the coarse fluxes):

$$\begin{aligned}\delta F^{s,\ell+1} &= -\triangle t^\ell\mathbf{F}^s(s^{n+\frac{1}{2},\ell}, \mathbf{v}_T^{n+\frac{1}{2},\ell})\cdot\hat{n} \quad \text{on} \quad \mathcal{P}(\partial\Omega^{\ell+1,*})\\ \delta F^{V,\ell+1} &= -\triangle t^\ell\mathbf{F}^V(V^{n+\frac{1}{2},\ell}, \mathbf{v}_T^{n+\frac{1}{2},\ell})\cdot\hat{n} \quad \text{on} \quad \mathcal{P}(\partial\Omega^{\ell+1,*}).\end{aligned}$$

If a coarser level exists, we increment the level ℓ flux registers (in this step, we are concerned with the level $(\ell-1)/\ \ell$ interface, so the level ℓ fluxes are the fine fluxes):

$$\begin{aligned}\delta F^{s,\ell} &= \delta F^{s,\ell} + \triangle t^{\ell} < \mathbf{F}^{s}(s^{n+\frac{1}{2},\ell}, \mathbf{v}_T^{n+\frac{1}{2},\ell}) \cdot \hat{n} > \quad \text{on} \quad \mathcal{P}(\partial\Omega^{\ell,*}) \\ \delta F^{V,\ell} &= \delta F^{V,\ell} + \triangle t^{\ell} < \mathbf{F}^{V}(V^{n+\frac{1}{2},\ell}, \mathbf{v}_T^{n+\frac{1}{2},\ell}) \cdot \hat{n} > \quad \text{on} \quad \mathcal{P}(\partial\Omega^{\ell,*}).\end{aligned}$$

5. Advance finer levels.

Next, we recursively advance the finer levels – we update level $\ell+1$ grids n_{ref}^{ℓ} times with $\triangle t^{\ell+1} = \frac{\triangle t^{\ell}}{n_{ref}^{\ell}}$.

6. Synchronize the current level with the finer levels.

After updating the finer levels, we synchronize the current level and finer levels by refluxing the saturation and freestream quantity on the current level:

$$\begin{aligned}s^{n+1,\ell} &= s^{n+1,\ell} + \tfrac{1}{\phi} D_R(\delta F^{s,\ell+1}) \\ V^{n+1,\ell} &= V^{n+1,\ell} + \tfrac{1}{\phi} D_R(\delta F^{V,\ell+1}).\end{aligned}$$

The last step in the algorithm is to average down the freestream quantity and the saturation from the finer levels to the current level; note that we use a volume-weighted average based on pore volume. We do not calculate velocities or pressures at the new time; any effort to compute these quantities is wasted, since they will change when the advected quantities are refluxed and averaged down.

3.3. Grid Generation

At the initial time step and at user-specified intervals, a new hierarchy of grids is created. This hierarchy of grids should have a high grid efficiency; grid efficiency is defined as the percentage of cells in refined grids that actually need to be refined. On the other hand, the grids need to be large and blocky to reduce the computational overhead at grid boundaries.

In this work, we use a two-step regridding algorithm. The first step is to tag the cells that need refinement; determining which cells to tag is the subject of this subsection. The second step is to create grids from these tagged cells using the clustering algorithm developed by Berger and Rigoutsos [9]. In their algorithm, the tagged cells are broken into rectangular blocks by using signature arrays and second derivatives of signature arrays to position the edges of grids. This procedure is repeated until the grids meet the user-specified grid efficiency.

In this work, we focus on different methods for tagging cells that need refinement. We use four different ways of tagging cells for refinement: 1) tagging user-specified regions, 2) tagging based on user-specified criteria, 3) tagging using an estimate of the truncation error, and 4) tagging using a flux-based error. Any combination of these four methods can be used to tag cells.

3.3.1. User-Specified Regions

The first method of tagging cells is for the user to specify the regions of refinement. In this case, we assume that the user knows a priori that certain regions of the domain will need to be refined for certain time periods. Since this is a static regridding criteria, it is generally not very useful for complicated time-dependent problems.

3.3.2. User-Defined Criteria

The second method of tagging cells is for the user to specify criteria for refinement. For example, the user may specify a cell is tagged if the pressure in that cell is greater than a user-specified pressure or if the difference in saturation between a cell and its neighbor is greater than a user-specified tolerance. In this case, the user must specify which flow variables are to be examined and what the tolerances are.

3.3.3. Local Truncation Error

The third method of tagging cells uses an estimate of the local truncation error (LTE). The LTE is computed, nondimensionalized, and compared to a user-specified tolerance. If the LTE is greater than the tolerance, then the cell is tagged for refinement. The major benefits of this method are that it is applicable to a wide variety of problems and it does not require the user to have any a priori knowledge of the problem.

We estimate the LTE by comparing the results of advancing the solution on two different levels. If we define a new level, ℓC, that consists of level ℓ grids coarsened by a factor of two and $L(s, \mathbf{v}_T)$ as the operator that advances s in time, then the estimate of the LTE, $e^{\ell C}_{LTE}$, can be written as:

$$e^{\ell C}_{LTE} =< L^{\ell}(s^{\ell}, \mathbf{v}^{\ell}_T) > -L^{\ell C}(Av(s^{\ell}), < \mathbf{v}^{\ell}_T >). \tag{18}$$

We note that computing $L(s, \mathbf{v}_T)$ is the same as computing $\frac{\partial s}{\partial t}$. On level ℓC, we use the velocity averaged down from level ℓ; thus, we avoid needing to solve an elliptic equation on level ℓC and the operator $L(s, \mathbf{v}_T)$ is simply:

$$L(s, \mathbf{v}_T) = -\frac{1}{\phi}\nabla \cdot \mathbf{F}. \tag{19}$$

To obtain the correct time-centering for $L(s, \mathbf{v}_T)$, we advance the solution from $t - \frac{1}{2}\triangle t^{\ell}$ to $t + \frac{1}{2}\triangle t^{\ell}$ on level ℓ and $t - \triangle t^{\ell}$ to $t + \triangle t^{\ell}$ on level ℓC.

Since our estimate of the LTE has units of $\frac{1}{T}$, we nondimensionalize it with a characteristic time, e^*, based on the maximum wave speed and the size of the media:

$$e^* = \frac{1}{D^*} max(|\frac{\partial \mathbf{F}}{\partial s}|\frac{1}{\phi}) \tag{20}$$

where D^* is the length of the media. Now, using the user-specified tolerance, tol, we tag cells where

$$\frac{|e_{LTE}|}{e^*} > tol.$$

At interfaces with coarser grids, we expect the scheme to lose accuracy due to coarse/fine interpolation; as a result, we expect to see a large error in the cells

neighboring the interface. If we use the normal tagging criteria, the interface cells will always be tagged and the fine grids will grow until the whole domain is refined. To avoid this problem, we simply copy the error from the neighboring cells that are not touching the interface. Since areas of high error tend to be in groups of cells rather than single cells, we expect this copying to work well. Additionally, since the error at the interface is on a set of one dimension less than the problem, it does not affect the global accuracy of the solution.

3.3.4. Flux-Based Error

The fourth method of tagging cells is an extension of the LTE-based tagging method that uses a flux-based error. In order for the LTE to be $O(h^2)$, there needs to be a cancellation of terms at cell edges. This cancellation occurs in interior cells, but does not occur at coarse/fine interfaces and physical boundaries; thus, we would expect the error to be larger at interfaces. To properly account for this larger error at the interface, we estimate the error by computing the error in the flux and then scaling it by a parameter based on the grid size. The implementation of this method is similar to the LTE-based method in Section 3.3.3 except that this method measures the error by comparing fluxes instead of the operator $L(s, \mathbf{v}_T)$.

In this method, we compute the time-centered fluxes on level ℓ and level ℓC as in the LTE-based tagging method. Then, for each edge of the cell, we calculate the flux difference – the difference between the average of the fine-grid fluxes and the coarse-grid flux. For the top edge of the cell, the flux difference, $\triangle F^T$, is defined as:

$$\triangle F^T = < F^{T,\ell C} > - F^{T,\ell}.$$

The left, right, and bottom edges are defined analogously.

The error for each cell, $e^{\ell C}_{FLUX}$, is defined as the maximum flux difference for any of that cell's edges multiplied by a surface to volume scaling factor SV:

$$e^{\ell C}_{FLUX} = max(\triangle F^T, \triangle F^B, \triangle F^R, \triangle F^L)(SV). \tag{21}$$

The SV factor is based on the size and shape of the grids. We define three measures of grid sizes – L^ℓ, LB^ℓ, and VOL^ℓ:

$$\begin{aligned} L^\ell &= \text{unduplicated perimeter of grids on level } \ell \\ LB^\ell &= \text{perimeter of grids on level } \ell \text{ that lie on physical boundary} \\ VOL^\ell &= \text{volume of grids on level } \ell. \end{aligned}$$

These measures of grid sizes have units; they are not simply dimensionless cell counts. With these definitions, we define the SV ratio at a cell edge on level ℓ as:

1. $SV = \frac{LB^\ell}{VOL^\ell}$, if the edge is at a physical boundary
2. $SV = \frac{L^\ell}{VOL^\ell}$, if the edge is not at a physical boundary and level $\ell + 1$ does not exist
3. $SV = \frac{L^{\ell+1}}{VOL^{\ell+1}}$, if the edge is not at a physical boundary and level $\ell + 1$ does exist.

The flux-based error has units of $\frac{1}{T}$, so we use the same nondimensionalization factor e^* as in the LTE-based tagging method (20). We would like to use the same

tolerance for the two methods, but the ranges on the errors are different – for a sample problem, the LTE-based method produced errors that ranged from 0 to 1.4, while flux-based method produced errors that ranged from 0 to 0.25. To estimate the difference in the ranges of the errors, we compare the flux-based error and the LTE-based method on a square grid with constant mesh spacing that covers the entire domain. For this sample problem with nx cells in the x-direction, the LTE-based method error will be approximately $\frac{4}{nx}$ bigger than the flux-based error. Instead of scaling the error estimate to account for this difference, we scale the tolerance for the flux-based error, tol':

$$tol' = tol\frac{4}{nx}$$

where tol is the tolerance used in LTE-based tagging method. Now, using the user-specified tolerance, tol, we tag cells where

$$\frac{|e_{FLUX}|}{e^*} > tol'.$$

4. RESULTS

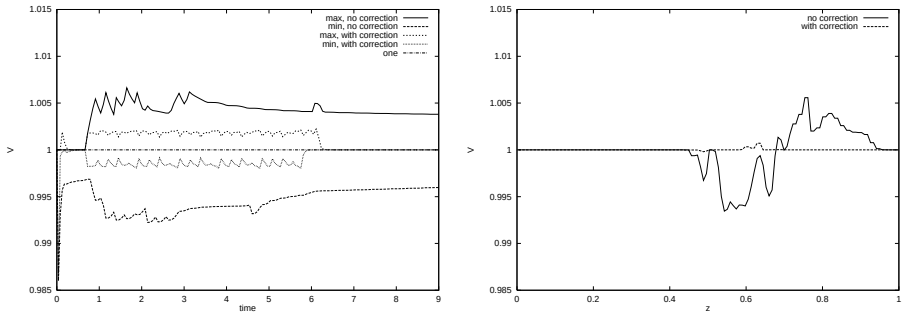
In this section, we describe the results of applying the adaptive algorithm to four test problems. The first test problem explores the effects of the freestream correction by showing the errors that can occur at coarse/fine interfaces and the effects of the correction algorithm. The second and third problems explore the use of adaptive algorithm on both smooth and discontinuous problems. The fourth problem demonstrates the various ways of tagging cells for refinement; we show three different methods for tagging cells and the grids that each method produces.

4.1. Problem 1: Freestream Preservation

In this test problem, we explore the effects of the freestream correction. We use the Cartesian coordinate system and a 64×64 base grid with a refinement ratio of 2. The domain is 1.0 by 1.0 and initially contains fluid with a saturation of 0.60; fluid is injected with a saturation of 0.70. We run two simulations – one using the freestream correction with $\eta = 0.99$ and the other without the correction.

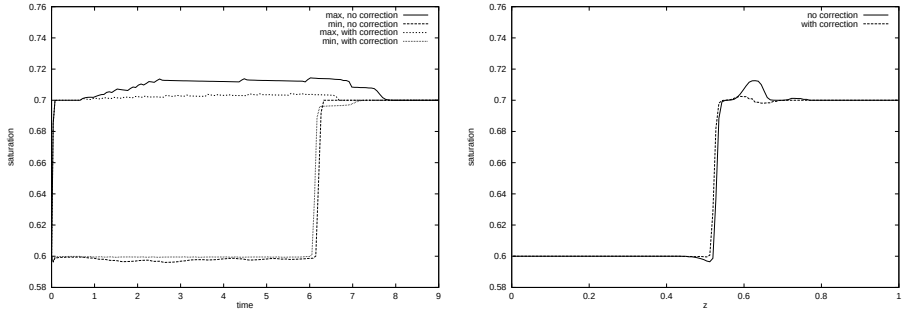
Figure 4(a) shows the maximum and minimum freestream quantity, V, within the domain as a function of time. Even with the correction, V is not identically one – this is a result of only using an approximate correction method. However, the error in V with the correction is 3-4 times smaller than without the correction. In addition, in the corrected case, V finally reaches 1.0 at time $t = 6.25$ – this corresponds to the saturation front reaching the outlet. On the other hand, in the uncorrected case, V never reaches 1.0.

Figure 4(b) shows a z-slice of the freestream quantity at time $t = 2.93$ for both the corrected and uncorrected cases. In the uncorrected case, the freestream errors accumulate; there are errors at every previous coarse/fine interface. On the other hand, in the corrected case, the only error is at the last coarse/fine interface. This indicates that the correction algorithm is succeeding in eliminating errors at interfaces from previous time steps.



(a) (b)

FIG. 4. Freestream quantity, V: (a) as a function of time, and (b) a vertical slice at time $t = 2.93$. The freestream correction significantly decreases the error in V, but it does not completely eliminate it.

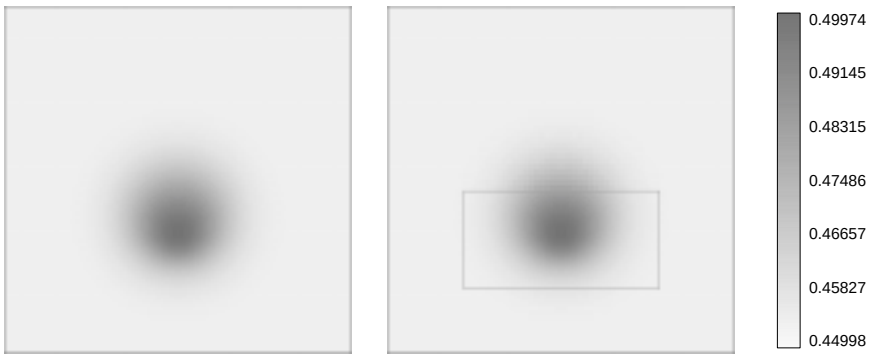


(a) (b)

FIG. 5. Saturation: (a) as a function of time, and (b) a vertical slice at time $t = 2.93$. Although the freestream correction does not eliminate the overshoot above 0.7, it does significantly decrease the amount of the overshoot.

Figure 5(a) shows the maximum and minimum saturation in the domain as a function of time. For this sample problem, the saturation should always be between 0.6 and 0.7. While both schemes have unphysical jumps above 0.7 and below 0.6, the corrected scheme has much smaller jumps. Again, this indicates that the scheme is only approximately correcting for the freestream problems. In addition, we note that it takes much longer for the uncorrected scheme to reach the correct steady state value of 0.7.

Figure 5(b) shows a z-slice of the saturation at time $t = 2.93$ for both the uncorrected and corrected cases. The most striking feature in this figure is the presence of an unphysical front. In the uncorrected case, the front has a maximum value of 0.71255 at $z = 0.63$, while in the corrected case, the front has been smoothed out over several rows of cells and has a maximum value of only 0.70256; thus, the correction has not eliminated the error, but it has significantly decreased the magnitude of the error.



(a) (b)

FIG. 6. Advection of a Gaussian distribution: (a) 128 x 128 grid, and (b) 64 × 64 grid with $n^0_{ref} = 2$.

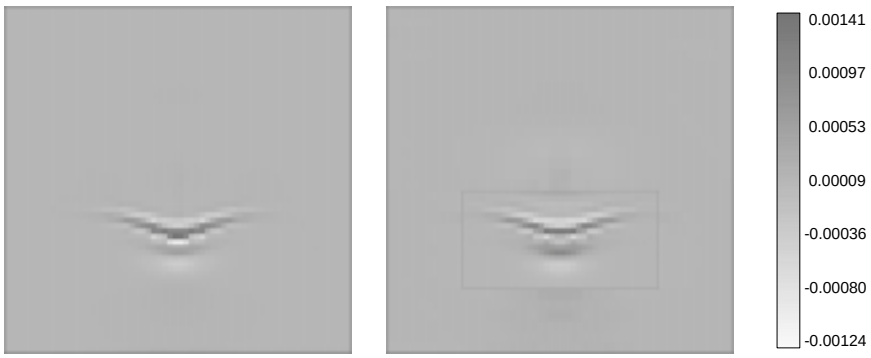
TABLE 1
Summary of results for the advection of a Gaussian distribution.

Base Grid	n^0_{ref}	n^1_{ref}	minimum saturation	maximum saturation	CPU time	cell count
64	-	-	0.44998	0.49921	464	81920
32	2	-	0.44998	0.49922	510	42176
128	-	-	0.44998	0.49974	3642	655360
64	2	-	0.44998	0.49974	2307	189056

4.2. Problem 2: Gaussian Distribution

In this test problem, we advect a Gaussian distribution of saturation in the Cartesian coordinate system. Figure 6(a) shows the saturation on a single 128 × 128 grid, while Figure 6(b) shows the saturation on an equivalent AMR grid - a 64 × 64 base grid with a refinement ratio of 2. These two figures are qualitatively similar.

Table 1 summarizes the results of these two simulations plus simulations on a single 64 × 64 grid and a 32 × 32 base grid with a refinement ratio of 2. The table shows that the maximum and minimum saturations on the equivalent grids are approximately equal. In addition, the table shows cell counts and CPU time for each of the simulations. These cell counts represent the number of grids cells that were advanced in time to obtain the solution; cell counts include cells which are covered by cells from finer grids. The table shows that AMR reduced the cell count by 49% for the effective 64 × 64 grid and by 71% for the effective 128 × 128 grid. The CPU time increased by 10% on the effective 64 × 64 grid, but decreased by 37% on the effective 128 × 128 grid; this indicates that there were not enough cells on the smaller effective grid to compensate for the overhead of AMR. Since cell counts are proportional to memory usage and CPU times measure computational effort, we can see the benefits of using adaptive refinement.



(a) (b)

FIG. 7. Local truncation error on equivalent grids: (a) 128 × 128 grid, and (b) 64 × 64 grid with $n^0_{ref} = 2$.

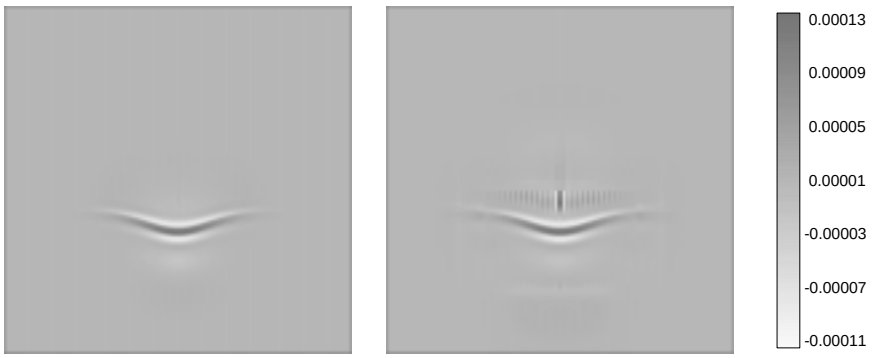
We can further illustrate that the adaptive algorithm is solving the problem correctly by examining two types of error in the solution – 1) the local truncation error and 2) the cumulative error in the solution. Local truncation error is an estimate of the error that has occurred over the last time step; it is computed by comparing the results of advancing the problem one time step on a coarse grid and one time step on a fine grid. Figure 7 shows the local truncation error on the 128 × 128 grid and the equivalent adaptive grid hierarchy. There is no noticeable increase in the local truncation error in the refined region of the adaptive grids. However, we do notice that there is an error at the coarse/fine interface due to the coarse/fine interpolation. Since the coarse/fine error is significantly smaller than the error in the refined region, it does not affect the quality of the solution.

We can estimate the cumulative error in the solution by comparing it with the solution on a 256 × 256 grid. This computation measures the total error in the solution; this is contrasted with the local truncation error which only measures the error in the solution over the last time step. Figure 8 shows the cumulative error in the solution on the 128 × 128 grid and the equivalent adaptive grid hierarchy. Again, we notice that the error in the refined region of the adaptive grid is the same as in the single-grid case and there are small regions of error at the coarse/fine interfaces. The cumulative error and the local truncation error are scaled differently, so it is not useful to compare their magnitudes, but they do produce qualitatively similar results.

4.3. Problem 3: Discontinuous Solution

In this test problem, we demonstrate the ability of the algorithm to solve problems with saturation fronts. The domain is 1.0 by 1.0 and initially contains fluid with a saturation of 0.45; at time $t = 0.0$, fluid is injected with a saturation of 0.70. The porosity varies across the width of the domain:

$$\phi(x) = 0.35 + 0.14(x - 0.5)\frac{e^{16.6|x-0.5|}}{e^{8.333}}.$$



(a) (b)

FIG. 8. Cumulative error on equivalent grids: (a) 128 × 128 grid, and (b) 64 × 64 grid with $n^0_{ref} = 2$.

TABLE 2
Summary of results for the discontinuous problem.

Base Grid	n^0_{ref}	n^1_{ref}	minimum saturation	maximum saturation	CPU time	cell count
32	-	-	0.44992	0.70011	209	31744
64	-	-	0.44988	0.70027	1484	237568
32	2	-	0.44996	0.70066	1269	100096
128	-	-	0.44986	0.70058	11497	1851392
64	2	-	0.44990	0.70076	6471	524864
32	4	-	0.44995	0.70058	5510	496384
32	2	2	0.44981	0.70081	4926	426624

We run this problem to time $t = 1.35$ on two series of equivalent grids. For the first series of problems, we use two grids with a fine resolution of 64 × 64: 1) a 64 × 64 grid and 2) a 32 × 32 base grid with a refinement ratio of 2. For the second series of problems, we use four grids with a fine resolution of 128 × 128: 1) a 128 × 128 grid, 2) a 64 × 64 base grid with a refinement ratio of 2, 3) a 32 × 32 base grid with a refinement ratio of 4, and 4) a 32 × 32 base grid with two levels of refinement and a refinement ratio of 2. The results of these four problems are shown in Figure 9. In both series of simulations, there is very little difference in the solution between equivalent grids.

Table 2 summarizes some of the results of these simulations. It shows that while the maximum and minimum saturations do vary slightly between equivalent grids, the error is still small in comparison with the error due to the numerical method. The cell count and CPU time columns indicate the benefits of refinement – in particular, we note that for the equivalent 128 × 128 grid, AMR reduced the cell counts by 72-77% and the CPU times by 44-57%.

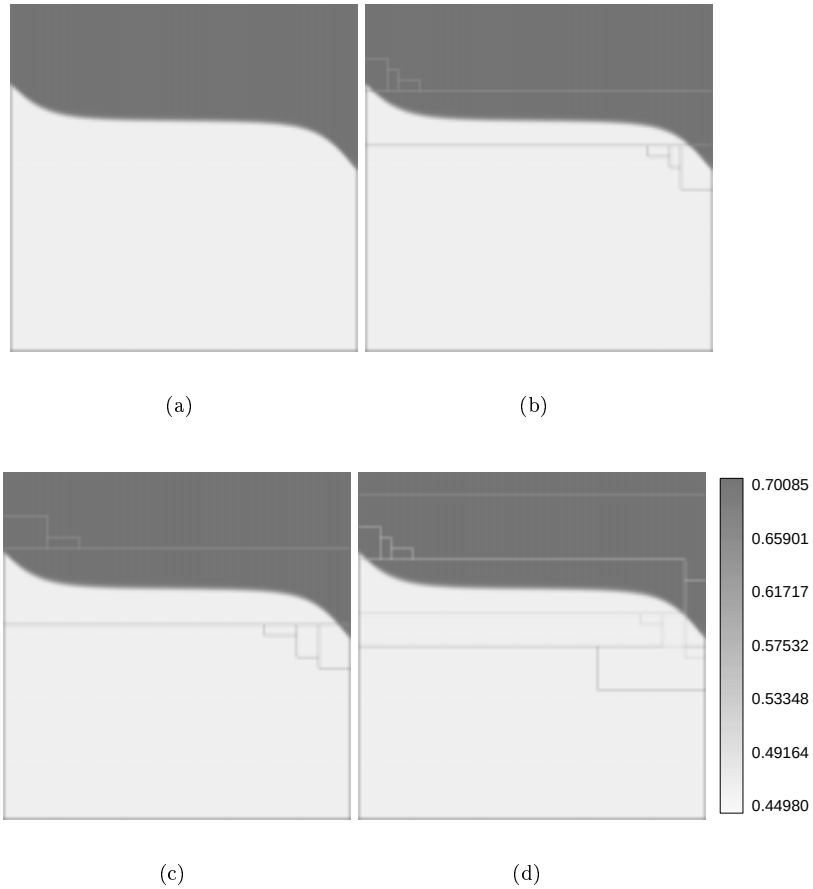


FIG. 9. Saturation front on equivalent grids: (a) 128 × 128 grid, (b) 64 × 64 grid with $n^0_{ref} = 2$, (c) 32 × 32 grid with $n^0_{ref} = 4$, and (d) 32 × 32 grid with $n^0_{ref} = 2$ and $n^1_{ref} = 2$.

4.4. Problem 4: Error Tagging

This test problem illustrates the use of three different methods of tagging cells for refinement – 1) tagging based on a user-specified criteria, 2) LTE-based tagging tagging, and 3) flux-based tagging. All three methods are applied to the same model problem – the discontinuous front moving through the porous medium (this is the same model problem that was discussed in Section 4.3). In all three cases, the solution and the grids are identical immediately before regridding, as shown in Figure 10(a). In addition, all three methods use a grid efficiency of 0.80.

The first method used to tag cells is the user-specified criteria method. We tag cells when the saturation changes by more than 0.1 between neighboring cells. This is a computationally inexpensive way to tag cells; however, it does require a priori knowledge of the flow field – we must know that saturation is the quantity we want to examine and that 0.1 is a large enough jump to require refinement. Figure 10(b) shows the error as determined by the saturation gradient and the grids it produces.

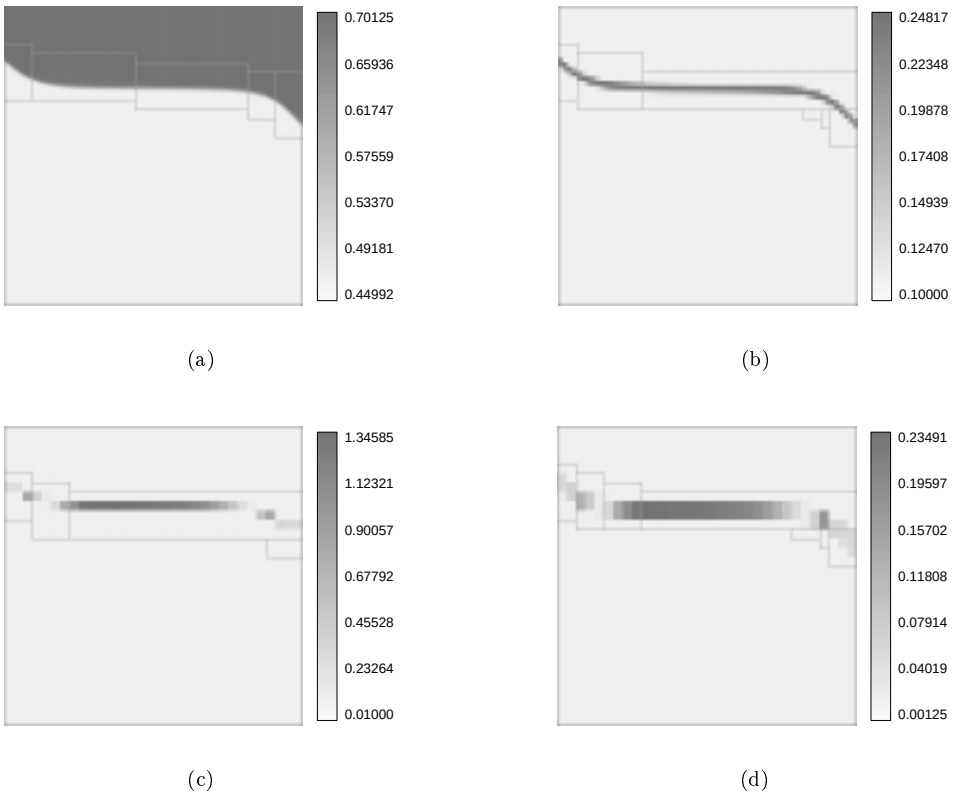


FIG. 10. The regridding process: (a) saturation before regridding, (b) error based on tagging the difference in saturation between neighboring cells and the resulting grids, (c) error based on operator and the resulting grids, and (d) error based on flux differencing and the resulting grids.

Since this method actually tags cells on level ℓ and the other methods tag cells on level ℓC, this method should tag fewer unnecessary cells.

The second method used to tag cells is the LTE-based tagging method. Figure 10(c) shows the estimate of the error using a tolerance of 0.01 and the grids it produces. The third method used to tag cells is the flux-based tagging method. Since we used a tolerance of 0.01 in the estimate of error example, we use a tolerance of $\frac{0.01}{8} = 0.00125$ in this example. Figure 10(d) shows the flux differencing error and the grids it produces. The scale of the plot shows that dividing by a factor of 8 does make the range of the flux-based error comparable to the range of the error in the estimate of LTE example.

All three methods of tagging cells were able to detect the discontinuity. As expected, each method tagged slightly different sets of cells and thus, produced slightly different grids. These slight differences do not affect the quality of the solution.

5. CONCLUSIONS

This paper described the extension of the single-grid code developed in [16] to an adaptive hierarchy of grids. The adaptive algorithm used subcycling to allow

for refinement in time, combined with a volume-discrepancy method to correct for freestream preservation problems at coarse/fine interfaces. Godunov's method and the elliptic solvers were modified to work on the AMR hierarchy. The elliptic solvers used a lagged correction scheme for coarse/fine boundary conditions to reduce the number of composite solves. In addition, we used several different methods to determine where patches of refinement were needed.

Results showed that the freestream correction is necessary at coarse/fine interfaces. Our freestream correction did not completely eliminate the freestream preservation problem, but it did significantly decrease the magnitude of the error. Simulations demonstrated that the adaptive code was able to reproduce single-grid results for both smooth and discontinuous problems; in addition, these results were obtained with less computational effort. Finally, several methods of tagging cells for regridding were demonstrated. Each method produced a slightly different set of grids, but these slight differences in grids did not affect the quality of the solution.

This adaptive code is meant as an initial step toward the design of a more complicated simulator of flow in porous media. One way to extend this work is to add more physical effects; these effects include multiple phases and components, heat and mass transfer, chemical reactions, and capillary pressure. Another way to extend this work is to add a more realistic geometry. This extension could include using a Cartesian grid to model heterogeneities in the flow or using a three-dimensional coordinate system.

APPENDIX: ELLIPTIC SOLVERS

In this appendix, we describe the use of two types solvers – level solvers and composite solvers. For each of these types, we describe two methods of solution – a multigrid based method and a biconjugate gradient stabilized method. We focus on the solution of $L(\phi) = \rho$, where the L operator is the elliptic operator defined in Section 3.1.1.

We define r^ℓ and e^ℓ as the residual and correction on level ℓ. In addition, we define $Smooth^\ell(e^\ell, r^\ell)$ as the operator that smoothes the solution on a level. In the standard multigrid algorithm, the smoothing operator is typically Gauss-Seidel with Red-Black ordering (GSRB). For our level and composite solvers, we use the $L^\ell(e^\ell, e^{\ell-1} = 0)$ operator (the level operator with all the coarse grid information set to zero). As a result, the only information that $Smooth^\ell$ requires about the other levels in the hierarchy is n^ℓ_{ref}.

A.1. LEVEL SOLVERS

A level solver on level ℓ solves the elliptic problem without knowledge of data from any other level; level ℓ is treated as both the finest and coarsest level of data. As a result, the only difference between a level solver and the standard multigrid solver are the grids. In a standard solver, each level consists of a single grid that covers the entire problem domain. In a level solver, each level may consist of several grids which generally do not cover the entire problem domain.

In this work, we use two types of level solvers – a multigrid-based algorithm and a biconjugate gradient stabilized method with a multigrid preconditioner. The multigrid level solver uses V-cycles with a conjugate gradient bottom solver. Since

a level solver does not include data from other levels, it uses the L^ℓ operator. The biconjugate gradient stabilized (BiCGStab) level solver is the standard BiCGStab solver with a dot product defined on a union of grids (see Barrett et al. [3] for a more detailed discussion of BiCGStab). The biconjugate gradient stabilized method requires about 3-5 times more memory than the multigrid-based method, but is generally faster when there are many small grids.

A.2. COMPOSITE SOLVERS

A composite solver on level ℓ_{base} solves on all grids finer than level ℓ_{base}. In other words, it starts at level ℓ_{max} and solves on each level until it reaches level ℓ_{base} and then uses a bottom solver. The solution is then interpolated back to level ℓ_{base}.

The composite solver has another significant feature – if possible, it computes multigrid coefficients using data from other levels in the grid hierarchy rather than simply averaging down the coefficients. For example, consider a grid hierarchy with a constant refinement ratio of two. If we coarsen the finest grid in the hierarchy within a multigrid solver, then the new multigrid level corresponds to level ℓ_{max-1} in the AMR hierarchy; as a result, we can use the saturations and other data on level ℓ_{max-1} to compute the multigrid coefficients on level ℓ_{max-1}, instead of averaging down the coefficients from level ℓ_{max}

A.2.1. Multigrid

The composite multigrid elliptic solver is based on the elliptic solver described by Martin and Cartwright [15]. The algorithm is written in residual-correction form and is summarized in Figure 1. The first step of the composite algorithm is to compute the residual on the finest level, $r^{\ell_{max}}$. Then, we save a copy of the solution, $\phi^{\ell,save}$, for later use. Next, we apply the smoother, $Smooth(e^\ell, r^\ell)$, to get an estimate of the correction on the current level, e^ℓ, and then update the solution with the correction. Then, we compute the residual on level $\ell-1$ based on whether the cell on level $\ell-1$ is covered by a grid on level ℓ:

$$r^{\ell-1} = \begin{cases} < r^\ell - L^\ell(e^\ell, e^{\ell-1}) > & \text{on } \mathcal{P}(\Omega^\ell) \\ \rho^{\ell-1} - L^\ell(\phi^{\ell-1}) & \text{on } \Omega^{\ell-1} - \mathcal{P}(\Omega^\ell). \end{cases}$$

This process of computing the correction and averaging down the residual is repeated until we reach level ℓ_{base}. We solve for $e^{\ell_{base}}$ by continuing the multigrid V-cycle; when we can no longer coarsen the grids, we use a conjugate gradient bottom solver. The level ℓ_{base} solution is then updated with the correction.

Then, we progress up the V-cycle. First, we use the $I^\ell_{\ell-1}$ operator to interpolate the level $\ell+1$ correction to level ℓ. However, the problem is no longer homogeneous, so we must modify the residual:

$$r^\ell = r^\ell - L^\ell(e^\ell, e^{\ell-1}).$$

We then smooth to solve for the correction to the correction, δe^ℓ and update the correction and the copy of the solution we saved. This process is repeated on progressively finer levels until we reach the finest level.

A.2.2. BiConjugate Gradient Stabilized

MG()
$r^{\ell_{max}} = \rho^{\ell_{max}} - L^{comp,\ell_{max}}(\phi^{\ell_{max}})$
do while $(iter < maxIter$ and $\|\mathrm{r}^{\ell_{\max}}\| < \mathrm{tol})$
iter++
AmrMG(ℓ_{max})
$r^{\ell_{max}} = \rho^{\ell_{max}} - L^{comp,\ell_{max}}(\phi^{\ell_{max}})$
end do
end MG

AmrMG (ℓ)
if $(\ell = \ell_{max})$
$r^{\ell} = \rho^{\ell} - L^{\ell}(\phi^{\ell}, \phi^{\ell-1})$
end if
for $\ell = \ell_{max}..\ell_{base}+1$,-1
$\phi^{\ell,save} = \phi^{\ell}$
$e^{\ell-1} = 0$
Smooth(e^{ℓ}, r^{ℓ})
$\phi^{\ell} = \phi^{\ell} + e^{\ell}$
$r^{\ell-1} = \begin{cases} < r^{\ell} - L^{\ell}(e^{\ell}, e^{\ell-1}) > & \text{on } \mathcal{P}(\Omega^{\ell}) \\ \rho^{\ell-1} - L^{\ell}(\phi^{\ell-1}) & \text{on } \Omega^{\ell-1} - \mathcal{P}(\Omega^{\ell}) \end{cases}$
end for
Solve$(e^{\ell_{base}}, r^{\ell_{base}})$
$\phi^{\ell_{base}} = \phi^{\ell_{base}} + e^{\ell_{base}}$
for $\ell = \ell_{base}+1..\ell_{max}$
$e^{\ell} = e^{\ell} + I^{\ell}_{\ell-1}(e^{\ell-1})$
$r^{\ell} = r^{\ell} - L^{\ell}(e^{\ell}, e^{\ell-1})$
$\delta e^{\ell} = 0$
Smooth$(\delta e^{\ell}, r^{\ell})$
$e^{\ell} = e^{\ell} + \delta e^{\ell}$
$\phi^{\ell} = \phi^{\ell,save} + e^{\ell}$
end for
end AmrMG

FIG. 1. Pseudocode for a composite solver using a multigrid-accelerated iterative method.

The composite BiCGStab elliptic solver is based on the elliptic solver of Bettencourt [10]. The algorithm for a composite BiCGStab solver is the same as the BiCGStab algorithm for a level solver except for three modifications due to the presence of a hierarchy of grids. First, the vector quantities are defined on each level from ℓ_{base} to ℓ_{max}; as a result, there is a loop over levels when each of these quantities is calculated. Second, the *Dot* operator becomes a composite operator, Dot^{comp}; we set all the data that is covered by a finer grid to zero, and then sum the dot products for each of the levels. Third, since this is a composite solve, we use the $L^{comp,\ell}$ operator instead of the L^{ℓ} operator.

ACKNOWLEDGMENT

We would like to thank Dan Martin and Matt Bettencourt for their help in developing and testing the AMR framework and elliptic solvers used in this work.

REFERENCES

1. G. Acs, S. Doleschall, and E. Farkas. General purpose compositional model. *Society of Petroleum Engineers Journal*, pages 543–552, August 1985.
2. A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell, and M. L. Welcome. A conservative adaptive projection method for the variable density incompressible Navier-Stokes equations. *Journal of Computational Physics*, 142:1–46, 1998.
3. R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, 1994.
4. J. Bear. *Dynamics of Fluids in Porous Media*. Dover Publications, 1972.
5. J. B. Bell, P. Colella, and J. A. Trangenstein. Higher order Godunov methods for general systems of conservation laws. *Journal of Computational Physics*, 82(2):362–397, 1989.
6. J. B. Bell, G. R. Shubin, and J. A. Trangenstein. A method for reducing numerical dispersion in two-phase black-oil reservoir simulation. *Journal of Computational Physics*, 65:71–106, 1986.
7. M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82(1):64–84, 1989.
8. M. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.
9. M. J. Berger and I. Rigoutsos. An algorithm for point clustering and grid generation. *IEEE Transactions Systems, Man and Cybernetics*, 21(5):1278–1286, 1991.
10. M. T. Bettencourt. *A Block Structured Adaptive Steady-State Solver for the Drift Diffusion Equations*. PhD thesis, UC Berkeley, 1998.
11. P. Colella. Multidimensional upwind methods for hyperbolic conservation laws. *Journal of Computational Physics*, 87:171–200, 1990.
12. R. E. Collins. *Flow of Fluids through Porous Materials*. Reinhold Publishing Corporation, 1961.
13. A. Gianetto and V. Specchia. Trickle-bed reactors: State of art and perspectives. *Chemical Engineering Science*, 47(13/14):3197–3213, 1992.
14. R. D. Hornung and J. A. Trangenstein. Adaptive mesh refinement and multilevel iteration for flow in porous media. *Journal of Computational Physics*, 136:522–545, 1997.
15. D. F. Martin and K. L. Cartwright. Solving Poisson's equation using adaptive mesh refinement. Technical Report UCB/ERL M96/66, UC Berkeley, 1996.
16. R. Propp, P. Colella, W. Y. Crutchfield, and M. Day. A numerical model for trickle bed reactors. *submitted to JCP*, 1999.
17. J. A. Trangenstein and J. B. Bell. Mathematical structure of the black-oil model for petroleum reservoir simulation. *SIAM Journal of Applied Math*, 49(3):749–783, 1989.
18. J. W. Watts. A compositional formulation of the pressure and saturation equations. In *7th SPE Symposium on Reservoir Simulation*, pages 113–122. SPE, November 1983.